



**Jet Propulsion Laboratory**  
California Institute of Technology

# F Prime: NASA Open Source Meets Small-Scale Flight Software

Michael Starch – SCaLE 17x – Pasadena, California – March – 2019

# About the Speaker

- Flight Software Engineer
- NASA Jet Propulsion Laboratory
- Open Source Contributor
- F' Missions and Improvements



# A Short Term Future (This Talk)

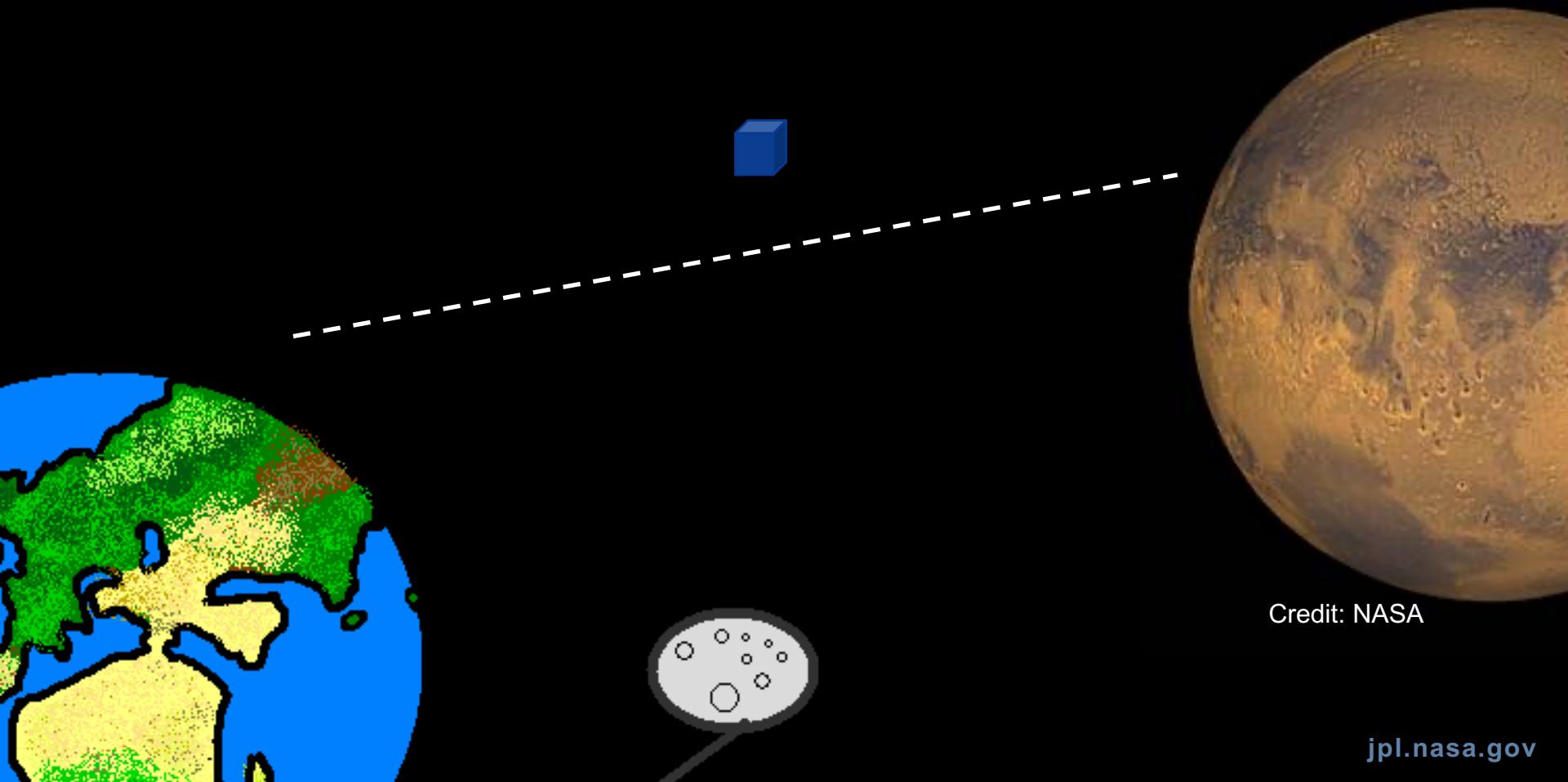
- Flight Software Primer
- An Introduction to F’
- F’ By Example
- F’ In Action
- F’ In the Real World
- F’ and The Future
- Questions – Anytime!

# Resources For Today's Talk

- NASA F' Landing Page:
  - <https://nasa.github.io/fprime>
- NASA F' Repository:
  - <https://github.com/nasa/fprime>
- Michael's Repo for Today's GPS Code
  - <https://github.com/LeStarch/fprime/tree/gps-application>

# Flight Software

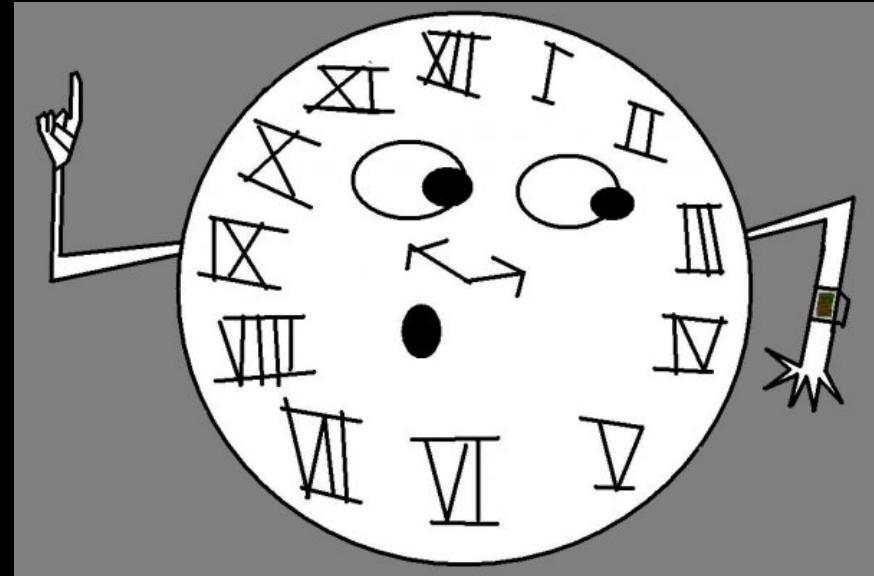
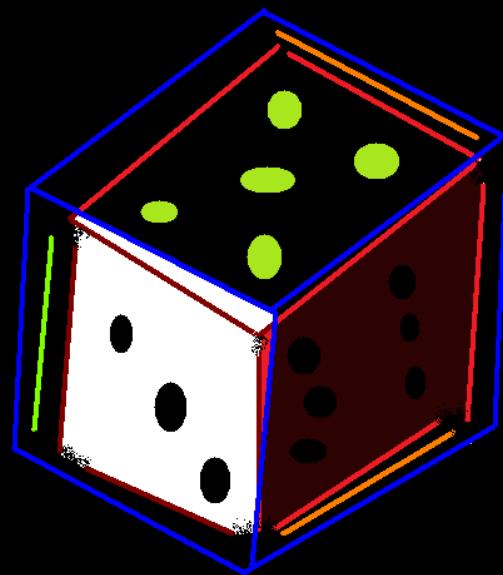
# Flight Software Primer



Credit: NASA

# Flight Software Primer

- High Reliability
- Deterministic
- Fault Tolerant
- Meets Deadlines
- Reviewable
- Collaborative
- Highly Tested



# Five Commandments of FSW

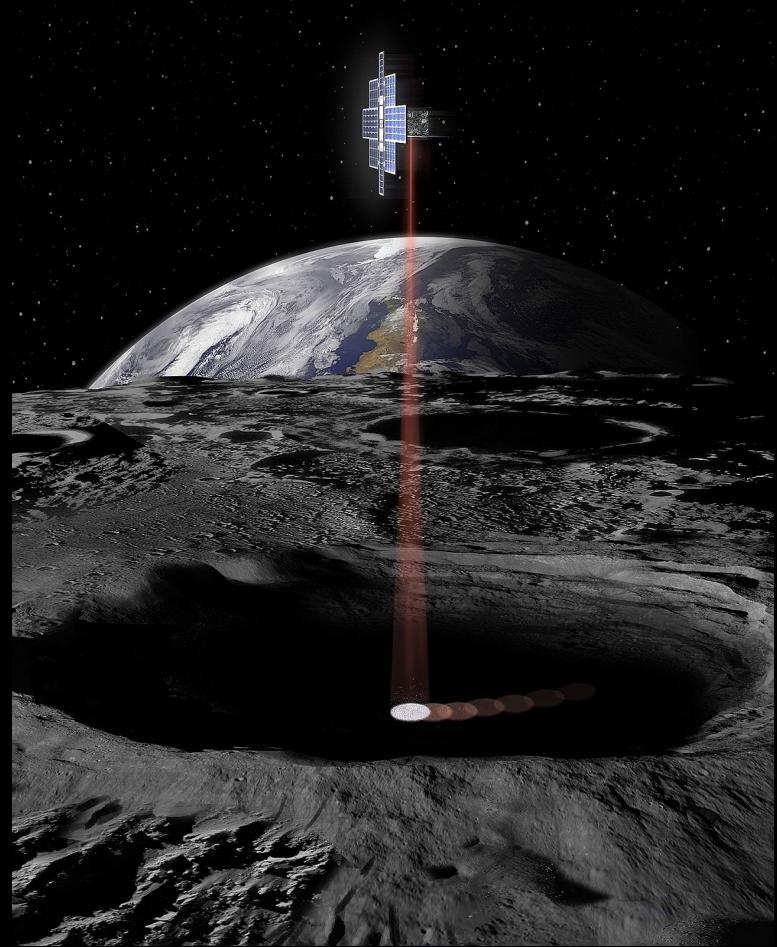
- Thou shalt not use new, nor malloc, nor any of their ilk after system initialization.
- Thou shalt assert thine assumptions.
- Thou shalt bound thine loops lest they run forever.
- Thou shalt despise recursion in all its forms.
- Thou shalt test thine code all the days of your life.



# F' Introduction

# What is F'?

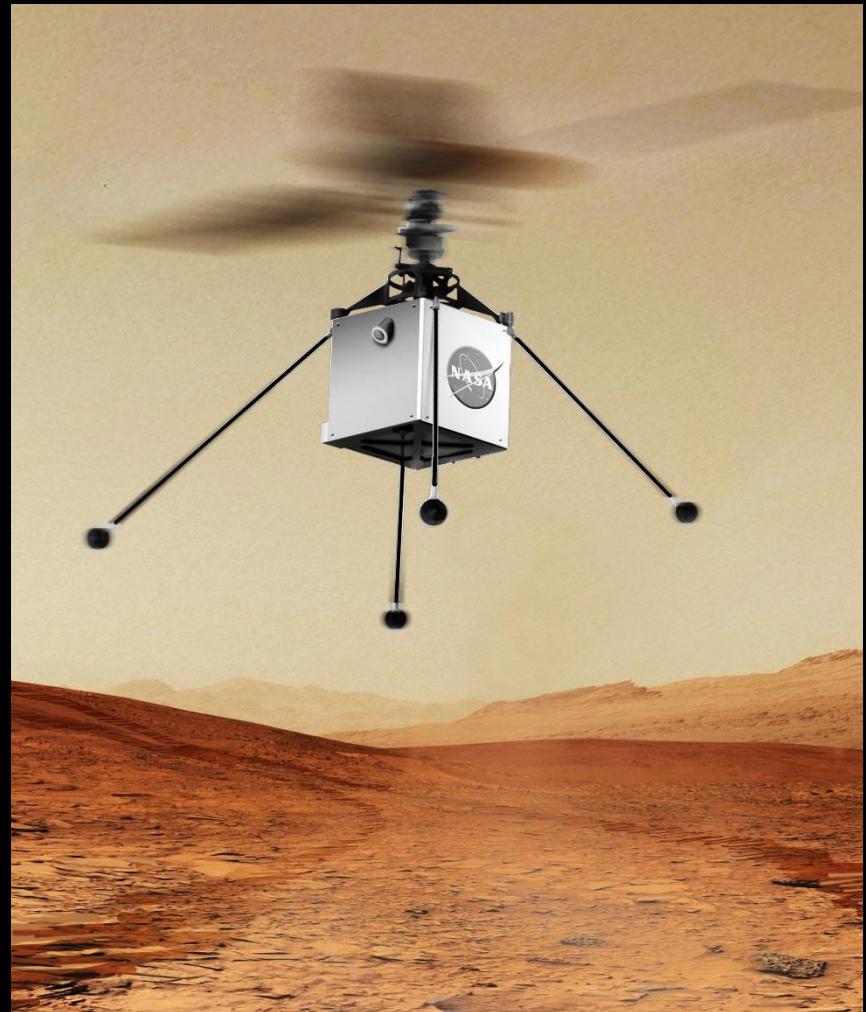
- Embedded Systems Framework
- Flight Software for Small-Scale Projects
  - Cubesats, Deployables, Instruments
- Provides:
  - Autocoder for Framework, Commands, Telemetry and Events
  - Standard Components and Abstractions
  - Modeling, Testing, and Basic Ground Support



Credit: NASA – 2015

# Why F'?

- Standard Components and Abstractions
- Flight Heritage
- Designed For Modularity and Reuse
- Compact Code, Efficient Implementation
- Scalability Features
- Unit Testing Framework



Credit: JPL/NASA – Artist's Concept

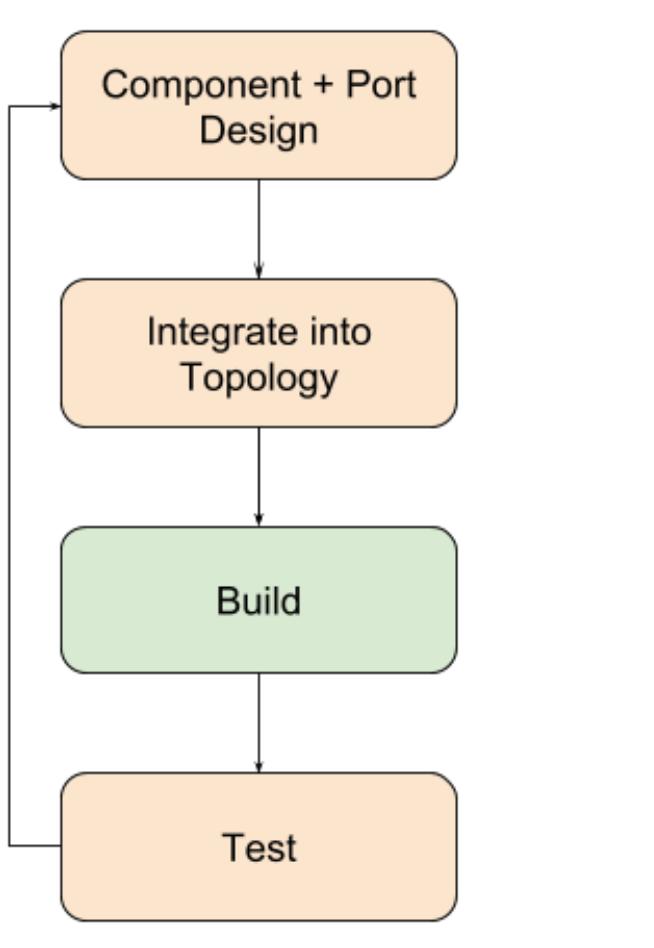
# A Crash Course in F' - Architecture

- Ports
  - Communication layer
  - Point-to-point between components
- Components
  - Primary building block of F' designs
  - Represents behavior of the system
- Topologies
  - Top-level system layout

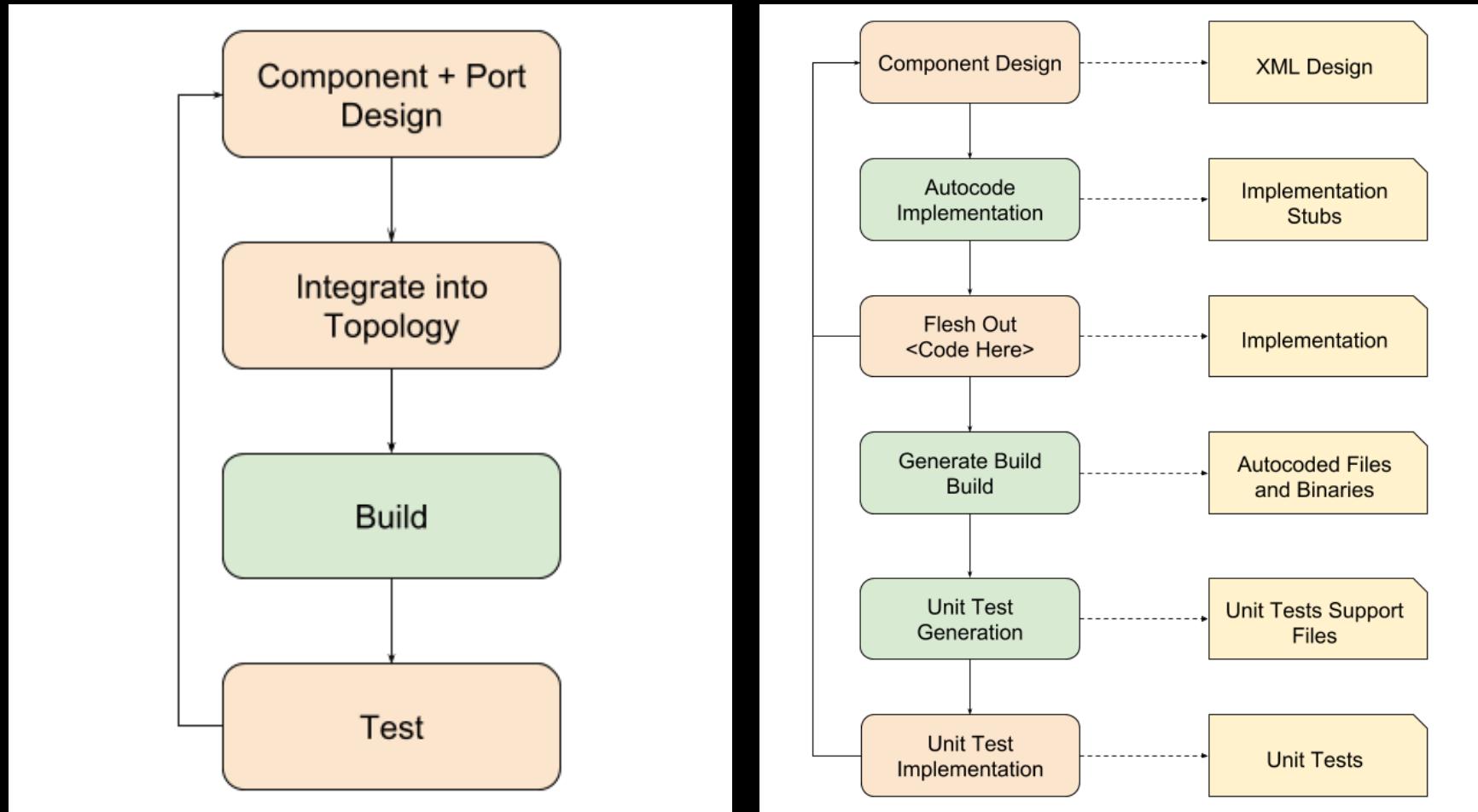
# A Crash Course in F' - Data

- Events:
  - “Log Messages”; something happened
  - ”ACTIVITY\_HI: Gps\_LockAquired: GPS lock acquired”
- Telemetry:
  - “Data Points”; state information
  - “Gps\_Latitude: 81.012”
- Commands:
  - Control and Input from ground
  - “Gps\_ReportLockStatus”

# F' Workflow



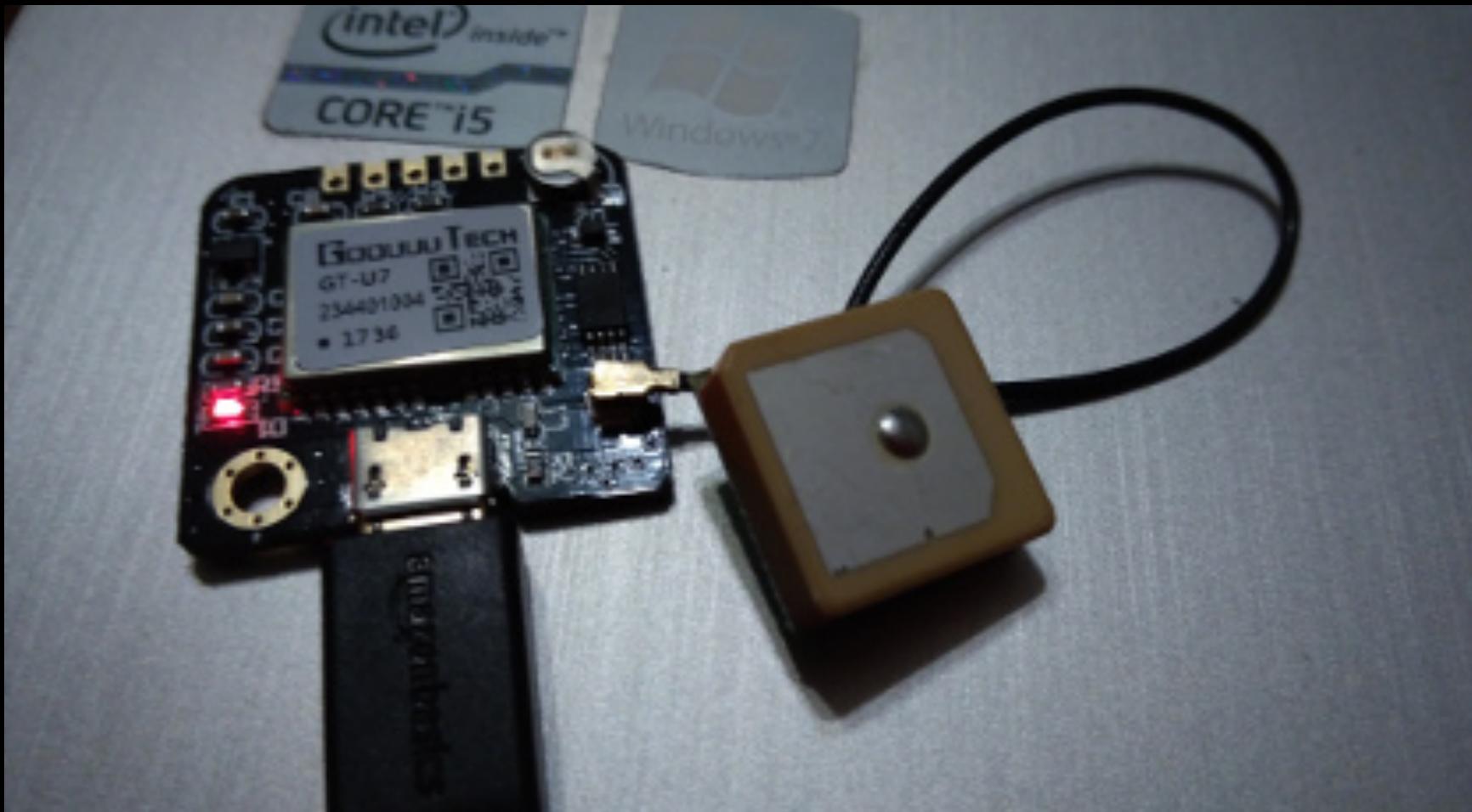
# F' Workflow



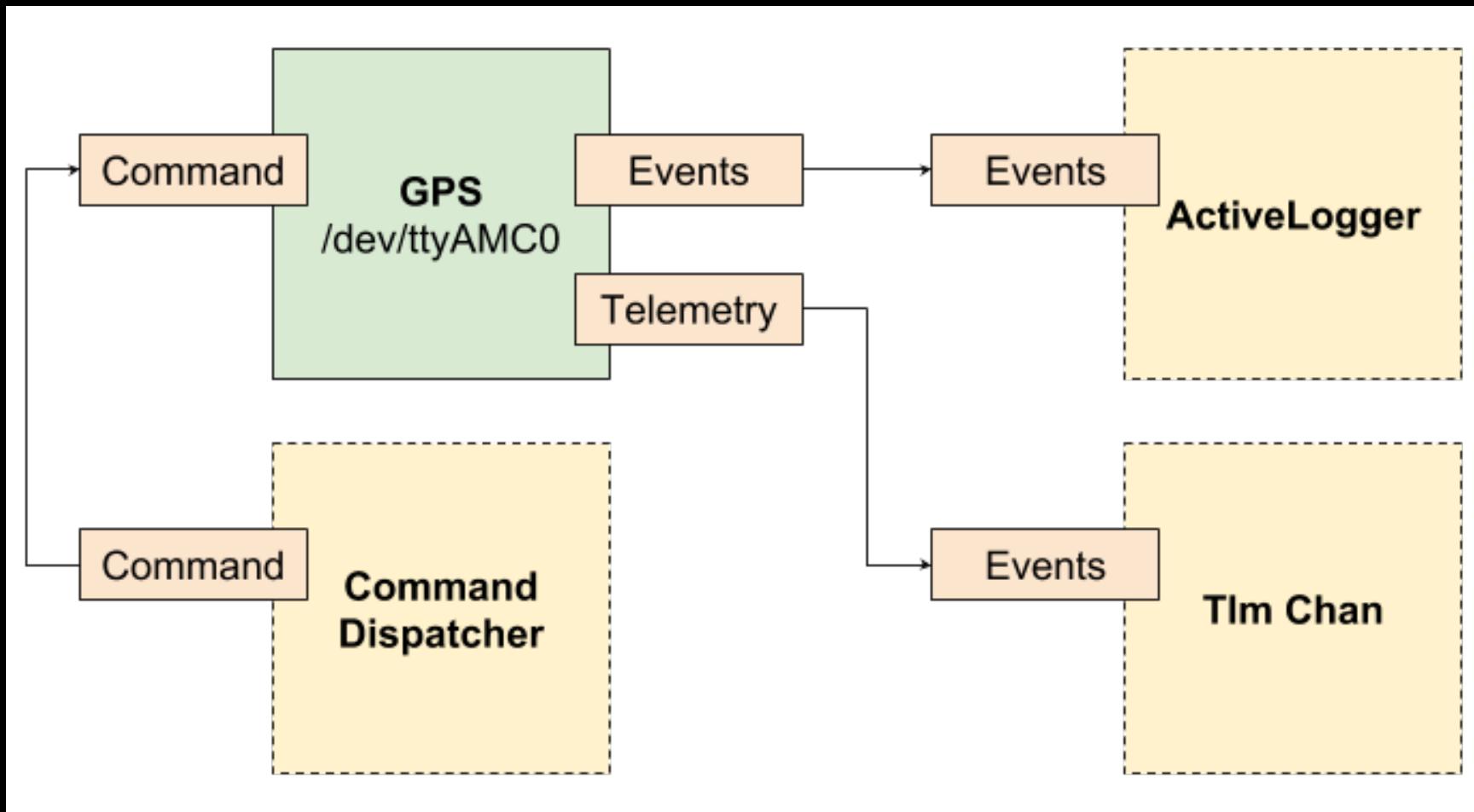


# F' By Example

# GPS Emitter in F'



# GPS Design in F'



# GPS Details

- Commands:
  - Report Lock State
- Telemetry:
  - Lat, Lon, Altitude
- Events:
  - Lock acquired; Lock Lost

# GPS Implementation - XML

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-model href="../../Autocoders/schema/ISF/component_schema.rng" type="application/xml" schematypens="http://relaxng.org/ns/structure/1.0"?
3
4  <component name="Gps" kind="active" namespace="GpsApp" modeler="true">
5
6      <import_port_type>Fw/Cmd/CmdRegPortAi.xml</import_port_type>
7      <import_port_type>Fw/Log/LogPortAi.xml</import_port_type>
8      <import_port_type>Fw/Cmd/CmdPortAi.xml</import_port_type>
9      <import_port_type>Fw/Sched/SchedPortAi.xml</import_port_type>
10     <import_port_type>Fw/Tlm/TlmPortAi.xml</import_port_type>
11     <import_port_type>Fw/Cmd/CmdResponsePortAi.xml</import_port_type>
12     <import_dictionary>GpsApp/Gps/Commands.xml</import_dictionary>
13     <import_dictionary>GpsApp/Gps/Telemetry.xml</import_dictionary>
14     <import_dictionary>GpsApp/Gps/Events.xml</import_dictionary>
15
16     <ports>
17
18         <port name="cmdRegOut" data_type="Fw::CmdReg" kind="output" role="CmdRegistration" max_number="1">
19             </port>
20
21         <port name="eventOut" data_type="Fw::Log" kind="output" role="LogEvent" max_number="1">
22             </port>
23
24         <port name="cmdIn" data_type="Fw::Cmd" kind="input" role="Cmd" max_number="1">
25             </port>
26
27         <port name="schedIn" data_type="Svc::Sched" kind="async_input" max_number="1">
28             </port>
29
30         <port name="tlmOut" data_type="Fw::Tlm" kind="output" role="Telemetry" max_number="1">
31             </port>
32
33         <port name="cmdResponseOut" data_type="Fw::CmdResponse" kind="output" role="CmdResponse" max_number="1">
34             </port>
35     </ports>
36
37 </component>
```

```
10    <telemetry>
11
12        <channel
13            id="0"
14            name="Gps_Latitude"
15            data_type="F32"
16            abbrev="GPS-0000"
17        >
18            <comment>The current latitude</comment>
19        </channel>
20
21        <channel
22            id="1"
23            name="Gps_Longitude"
24            data_type="F32"
25            abbrev="GPS-0001"
26        >
27            <comment>The current longitude</comment>
28        </channel>
29
30        <channel
31            id="2"
32            name="Gps_Altitude"
33            data_type="F32"
34            abbrev="GPS-0002"
35        >
36            <comment>The current altitude</comment>
37        </channel>
38
39        <channel
40            id="3"
41            name="Gps_Count"
42            data_type="U32"
43            abbrev="GPS-0003"
44        >
45            <comment>The current number of satellites</comment>
46        </channel>
47
48    </telemetry>
```

# GPS Implementation

```
98     // Step 0: schedIn
99
// By implementing this "handler" we can respond to the 1HZ call allowing
101 // us to read the GPS UART every 1 second.
102 void GpsComponentImpl ::

103     schedIn_handler(
104         const NATIVE_INT_TYPE portNum,
105         NATIVE_UINT_TYPE context
106     )
107 {
108     int status = 0;
109     float lat = 0.0f, lon = 0.0f;
110     GpsPacket packet;
111     char buffer[1024];
112     char* pointer = buffer;
113
//During each cycle, attempt to setup if not setup
114     //Step 1: setup
115     // Each second, we should ensure that the UART is initialized
116     // and if not, we should try to initialize it again.
117     setup();
118     if (!m_setup) {
119         return;
120     }
121
//Then receive data from the GPS. Should block until available
122 //and thus, this module should not be driven at a rate faster than 1HZ
123 //Step 2: read the UART
124     // Read the GPS message from the UART
125     ssize_t size = read(m_fh, buffer, sizeof(buffer));
126     if (size <= 0) {
127         std::cout << "[ERROR] Failed to read from UART with: " << size << std::endl;
128         return;
129     }
130
//Look for a recognized GPS location packet and parse it
131 //Step 3:
132     // Parse the GPS message from the UART (looking for $GPPA messages)
133     for (U32 i = 0; i < sizeof(buffer) - 6; i++) {
134         status = sscanf(pointer, "$GPGGA,%f,%f,%f,%f,%u,%f,%f",
135                         &packet.utcTime, &packet.dmNS, &packet.northSouth,
136                         &packet.dmEW, &packet.eastWest, &packet.lock,
137                         &packet.count, &packet.filler, &packet.altitude);
138
//Break when all GPS items are found
139     if (status == 9) {
140         break;
141     }
142 }
```

```
142     pointer = pointer + 1;
143 }
144
//If we failed to find the packet, or failed to extract data then return
145 if (status != 9) {
146     std::cout << "[ERROR] GPS parsing failed: " << status << std::endl;
147     return;
148 }
149
//GPS packet locations are of the form: ddmm.mmmm
150 //We will convert to lat/lon in degrees only before downlinking
151 //Latitude degrees, add on minutes (converted to degrees), multiply by direction
152 lat = (U32)(packet.dmNS/100.0f);
153 lat = lat + ((packet.dmNS - (lat * 100.0f))/60.0f);
154 lat = lat * ((packet.northSouth == 'N') ? 1 : -1);
155
//Longitude degrees, add on minutes (converted to degrees), multiply by direction
156 lon = (U32)(packet.dmEW/100.0f);
157 lon = lon + ((packet.dmEW - (lon * 100.0f))/60.0f);
158 lon = lon * ((packet.eastWest == 'E') ? 1 : -1);
159
//Step 4: downlink
160
// Call the downlink functions to send down data
161 std::cout << "[INFO] Current lat, lon: (" << lat << "," << lon << ")" << std::endl;
162 tlmWrite_Gps_Latitude(lat);
163 tlmWrite_Gps_Longitude(lon);
164 tlmWrite_Gps_Altitude(packet.altitude);
165 tlmWrite_Gps_Count(packet.count);
166
//Lock status update only if changed
167 //Step 7: note changed lock status
168 // Emit an event if the lock has been aquired, or lost
169 if (packet.lock == 0 && m_locked) {
170     m_locked = false;
171     log_WARNING_HI_Gps_LockLost();
172 } else if (packet.lock == 1 && !m_locked) {
173     m_locked = true;
174     log_ACTIVITY_HI_Gps_LockAquired();
175 }
176
177
178 // -----
179 // Command handler implementations
180 // -----
181 //Step 9: respond to status command
182 //
183 // When a status command is received, respond by emitting the
184 // current lock status as an Event.
185 void GpsComponentImpl ::
```



# F' In Action

# Example With PI: Prompt and GSE





# Real World F'

# F' for Full-Scale Missions

- Modeling Software
  - MagicDraw (UML/SysML) F' Plugin
  - MagicDraw is not OpenSource
- Unit Tests, Continuous Integration, and Testing
  - Jenkins and Docker
  - Multiple Deployments
  - Integration Tests
- Operations
  - F' and OS Integration
  - Communications
  - Ground System

# F' Best Practices and Tips

- Rate Groups for Deterministic Timing
- Parameter DB for Non-Volatile Behavior Affecting Values
- Hub Pattern for Off-Device Communication
- Fatal Handling and Health





# The Future

# F' and OpenSource



Credit: NASA/KSC

# F' and The Future





**Jet Propulsion Laboratory**  
California Institute of Technology

---

[jpl.nasa.gov](http://jpl.nasa.gov)